Optimal Task Allocation for Time-Varying Edge Computing Systems with Split DNNs

Davide Callegaro, Yoshitomo Matsubara and Marco Levorato
Department of Computer Science, University of California, Irvine
e-mail: {dcallega,yoshitom,levorato}@uci.edu

***Abstract*—Many modern applications rely on complex machine learning algorithms, such as Deep Neural Networks (DNNs), to analyze images. However, both mobile and edge computing strategies may fail to provide satisfactory performance in some parameter regions. To mitigate this issue, the research community recently proposed methods to split the execution of DNNs to optimize the balance between computing load allocation and channel usage. Building on this set of results, this paper presents an optimization framework that enables the dynamic control of how images are processed in mobile device-edge server systems. The system is modeled as a Markov process, and a Linear Fractional Program is defined to identify the optimal stationary state-action distribution minimizing the overall average inference time under a constraint on the number of discarded images. Results indicate the advantage of using a dynamic control strategy with respect to available fixed strategies.**

I. Introduction

Many modern mobile applications rely on complex machine learning algorithms, such as Deep Neural Networks (DNNs), to analyze images and extract information on their content. The high computational complexity of these algorithms clashes with the constrained computing and energy resources available to mobile devices. To address this issue, the research community proposed two main approaches: (*i*) reducing the complexity of the DNN models to fit within the constraints of the mobile device [1], and (*ii*) offloading the computing task to more powerful computers, such as edge servers [2]. On the one hand, the former option inevitably results in some degradation of the DNN output with respect to full models. On the other hand, edge computing necessitates the transfer of – possibly high resolution – images over wireless links. The instability of wireless links, and network load patterns in general, may degrade the performance of this strategy.

A recent trend of contributions proposed splitting the execution of DNNs models between the mobile device and edge server [3]–[7]. The idea is to divide DNN models into head and tail portions, which are executed at the mobile device and edge server, respectively. The channel, then, transports the output tensor of the head model to the edge server. Unfortunately, the structure of DNN models for vision task does not allow effective splitting, as they typically concentrate most computational complexity in the early layers, where they also tend to amplify the input size. Intuitively, splitting such architectures would result in an excessive computation load to the mobile device, as well as no advantage in terms of channel load. Some split DNN approaches, then, introduce a bottleneck layer early in the DNN structure to compress the input image into a small tensor, thus mitigating channel impairments that are the main source of performance degradation in edge computing-based systems [4]–[7].

In this paper, we take as a starting point the splitting approach we presented in [5], where the modification of the architecture was paired with a specific training strategy - Knowledge Distillation - applied to the first section of the model. Knowledge Distillation trains that portion of the modified model – which contains the bottleneck – to mimic the output of the original section of the model. The model is then split at the bottleneck to achieve compression. This approach showed some important advantages, including the ability to generate small bottlenecks without sacrificing overall accuracy even in complex vision tasks, such as classification on the ImageNet dataset.

However, splitting is of course optimal only in some regions of parameters describing the channel capacity and the characteristics and state of the edge server (*e.g.*, task queue length and computing capacity). In general, the three main options, local computing, edge computing and split computing, may be optimal in different conditions. As the system state evolves over time due to channel and queueing dynamics, a scheduling problem arises, where the mobile device needs to determine which one of the three options to choose. Intuitively, as tasks may accumulate, the decision needs to be optimal considering the statistics of the future system's state.

In this paper, we present a scheduling problem determining how images periodically produced by a mobile device are processed. We consider a system including a sensor generating images at the mobile device, processing units at the mobile device and edge server, a time-varying communication channel, and a selector deciding how each image is processed. As task flow may exceed the capabilities of the processing units and communication channel, we include in the system finite buffers to accumulate tasks to be completed and data to be transmitted. Notably, the presence of buffers, a crucial components of real-world systems, induces temporal correlation, where the choices of the selector at a given time influence the distribution of future system's states.

We model the system as a Markov process, and formulate an optimization problem whose objective is to minimize the average time between image capture and the availability of the analysis outcome and the number of images rejected by the buffers. The problem is mapped to a Linear Fractional

Program (LFP), whose optimization variables are state-action stationary frequencies.

The rest of the paper is organized as follows. In Section II, we provide an overview of prior work, and briefly explain the split DNN technique we proposed in [5]. Sections III and IV describe the system and the Markov process used to model its dynamics. Section V formulates the optimization problem and presents the resolution methods. Results are shown and discussed in Section VI, and Section VII concludes the paper.

II. Distilled Split DNN Models

There are several methods to make DNN models deployable such as training lightweight models [1,8], model compression and pruning [9,10]. Such approaches, however, often experience significantly degraded accuracy of the model predictions and/or require many iterations of complex operations in training and optimization.

Kang *et al.* [3] and Jeong *et al.* [11] propose to simply split DNN models in an edge computing scenario. However, such approaches are not well motivated, as many state-of-the-art DNN models do not present bottlenecks – that is, layers with few nodes – in the early layers. As a result, splitting is often suboptimal compared to local or edge computing from a point of view of total inference time. Recent studies attempt to introduce bottlenecks by modifying the architecture and training of the models [4]–[7].

For the sake of completeness, we briefly summarize here the split DNN technique we proposed in [5]. As mentioned in the introduction, the core idea is to introduce a bottleneck layer, that is, a layer composed of few nodes, in the early stages of the model. This enables to achieve in-network compression of the input while limiting the computing load assigned to the mobile device. We focus on complex DNN models: DenseNet-169, -201 [12], ResNet-152 and Inception-v3 [13], where we first modified the models to introduce such bottlenecks, and retrained the altered model using a technique called head network distillation [5]. The technique stems from *knowledge distillation* [14], a procedure used to train a smaller (student) model to mimic a bigger (teacher) model's output. Interestingly, it is reported that student models trained to mimic the teacher model often outperform equivalent models trained using a vanilla method. Since our focus is on introducing bottlenecks to the early layers of the pretrained model, we only train the head portion of the altered model to reduce training time.

Table I summarizes the head-distilled models' accuracy and bottleneck tensor size scaled by the input tensor size ($3 \times 299 \times 299$ for Inception-v3 and $3 \times 224 \times 224$ for others) as reported in our previous work [5]. With small test accuracy loss, our introduced bottlenecks reduce the size of data to be transferred to the edge computer by approximately 98% compared to the input tensor size. The reduction in network payload corresponds to a reduced communication delay withe respect to transmitting the input data, as shown in Section VI.

Table I: Classification performance of head-distilled (student) models with bottlenecks [5]

Altered model	DenseNet-169	DenseNet-201	ResNet-152	Inception-v3
Test accuracy [%]	83.3 (-1.2)	84.1 (-1.1)	83.2 (-1.1)	85.7 (-0.8)
Data size [%]	1.68	1.68	1.68	1.53

* The numbers in brackets indicate difference in accuracy from the original (teacher) models.

III. System Description

We consider a system composed of a mobile device (MD) and an edge server (ES). The overall objective of the system is to analyze images acquired by the MD in the shortest possible time and with the highest possible accuracy with the assistance of the ES. We consider a specific family of vision tasks, that is, image classification. In this work, we focus on a specific classification model being used to analyze all the images produced by the MD.

We denote the total time lapse from image acquisition to the availability of the output as T. To minimize T, the MD has three options:

Local Computing: the MD executes the DNN model using its own resources.

Edge Computing: The MD transmits the full image to the ES, which executes the model and transmits the outcome to the MD.

Split Computing: The MD executes the head model, transmits over the wireless channel the output tensor to the ES, which executes the tail model and sends back the outcome to the MD.

As explained in [5], the three different choices are optimal in different regions of parameters describing the computing capacity of the MD and ES, as well as the capacity of the wireless channel connecting them.

Herein, we develop a technique to allow the MD to dynamically select the best option in response to the system state. As both computing and communication tasks may accumulate within the system, we describe the latter as the concatenation of queues illustrated in Fig. 1. The MD acquires images, which are forwarded to a selector to determine which of local, edge and split computing is used. In the first case, the task is forwarded to the task buffer of the MD, and eventually executed by the embedded MD's processor. In the second case, the full input image is forwarded to the communication buffer and eventually delivered over the communication channel to the ES task buffer for processing. In the third case, the image is sent to the local task buffer, but only the head portion of the model is executed by the embedded processor, which then forwards the tensor to the communication buffer for transmission to the ES task buffer. The ES then executes the tail portion. Note that in the edge and split computing options, the model output needs to be transmitted back to the MD. The size of the MD task buffer, ES task buffer and communication buffer are denoted with N_{md}, N_{es} and N_{comm}, respectively. We adopt a First-In First-Out service model.

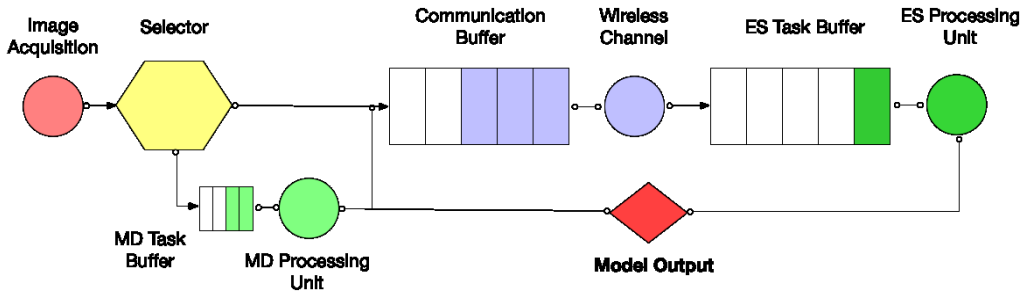

Figure 1: Schematics of the system considered in this paper.

The total delay T_i of image i is the sum of many components, whose value depends on the computing capacity of the MD and ES – which here is assumed fixed – as well as the current channel capacity and the state of the buffers, which vary over time. In order to minimize the average delay, the selector, then, inevitably needs to implement a policy capable of reacting to the dynamics of the system's state.

IV. Stochastic Model

In this section, we characterize the state space and dynamics of the stochastic process associated with the system described earlier. We note that in the following we use capital and lowercase letters to denote random variables and their values, respectively.

A. State Space

We define the state space of the system as the vector

$$\mathbf{s} = [c, \mathbf{b}_{\text{md}}, \mathbf{b}_{\text{comm}}, \mathbf{b}_{\text{es}}], \tag{1}$$

where c is the state of the wireless channel, and $\mathbf{b}_{\text{md}}$, $\mathbf{b}_{\text{comm}}$, and $\mathbf{b}_{\text{es}}$ are vectors describing the state of the MD task buffer, communication buffer and ES task buffer, respectively. The state of the channel corresponds to the Signal-to-Noise-Ratio (SNR) experienced by the link. We quantize the SNR to define C transmission rates, obtained using a capacity model, that is

$$\Psi(c)=W\log_2(1+\text{SNR}_c), \tag{2}$$

where SNR_c is the SNR associated with channel state $c\in\{1,\dots,C\}$, and W is the channel bandwidth.

The vector $\mathbf{b}_{\text{md}} = \{a_k\}_{1,\dots,N_{\text{md}}}$ contains N_{md} elements each of those is associated with a slot in the MD task buffer. The variables a_k lie in the set $\{0, \text{full}, \text{split}\}$, whose elements respectively correspond to an empty slot, a slot containing an input image to be processed using the original full model, and an image to be processed using the head model only. The buffer is organized to rank tasks in order of arrival, that is, the oldest task is in the first position, the second oldest on the second and so on, and empty slots are at the vector end. We define the functions

$$Z_{\text{md}}(\mathbf{z}) = z, \quad F_{\text{md}}(\mathbf{z}) = a_1, \tag{3}$$

where z is the number of empty slots and a_1 is the first element in the vector $\mathbf{b}_{\text{md}}$ within the state $\mathbf{z}$. We define analogous functions extracting the same quantities from $\mathbf{b}_{\text{es}}$ and $\mathbf{b}_{\text{comm}}$, whose definition is analogous. The elements a_k of $\mathbf{b}_{\text{comm}}$ are associated with empty slots (a_k=0), input images (a_k=full), and tensors (a_k=split) to be delivered to the ES buffer for processing. The elements a_k of $\mathbf{b}_{\text{es}}$ correspond to empty slots (a_k=0), input images to be fully processed by the ES (a_k=full), and tensors (a_k=split) that are inputs to the tail DNN model. On these models, we define similar functions as in Eq. 3.

In addition to the system state, we define the decision variable $u \in \{\text{lc}, \text{ec}, \text{split}\}$. The components of the decision space correspond to a task being fully executed locally at the MD, being fully offloaded to the ES, and split computing.

B. System Dynamics

In order to analyze the system and locate the optimal selection policy, we make some assumptions that are common in queueing system analysis. Specifically, we assume that the image interarrival time at the MD, the data transfer time, and the task execution time are exponentially distributed random variables. The distributions are centered on values extracted from the real-world experiments reported in [5].

We define, then, the following parameters

- λ as the arrival rate of images (that is, $1/\lambda$ is the average inter-capture time of images),
- γ_{full} and γ_{head} as the execution rate of the full and head DNN models at the MD, respectively.
- ρ_{full} and ρ_{tail} as the execution rate of the full and tail DNN models at the ES, respectively.
- $\nu_{(\text{i,in})}$ and $\nu_{(\text{i,head})}$ as the transmission rate (*i.e.*, the channel service rate) of full images and output tensors, respectively. The parameters are computed as the channel capacity of that state divided by the data size of the input image/tensor.
- Finally, we assume a *jump* model for the channel, where the channel state switches from state i to state j with probability ϕ_{ij} after an exponentially distributed time with parameter ω.

We emphasize that the service rate of the MD and ES task buffers depends on the nature of the task being executed (oldest in the buffer), which can be either the execution of the full DNN model, or the execution of the head (MD) and tail (ES) DNN models. Similarly, the channel service rate depends on the tag associated with the oldest data chunk in the buffer,

but also on the current channel capacity (described by the element c in the overall state vector).

Under the assumption that the service rates are exponentially distributed as defined above, the system dynamics can be described as a stationary Semi-Markov process $\{S_t\}$ with a finite state space as described in Section IV-A, where $t{=}1, 2, \ldots$. Different from plain Markov processes, in Semi-Markov processes the permanence time in each state is a random variable. The discrete temporal index t, then, refers to time instants right after a state change. We remark that as the timing of all events in the system is determined by exponentially distributed random variables, then the probability that a particular event is the next is computed as the probability that the corresponding random variable is the smallest. Moreover, again due to the exponential nature of inter-event time, the residual time of the variables upon the occurrence of an event has the same distribution. Importantly, the overall inter-event time is distributed as the minimum set of exponentially distributed variables.

Assuming a recurrent Markov chain, the long-term dynamics of the process are fully defined by the transition probabilities

$$P_u(ij) = \Pr\{S_{t+1}{=}j \mid S_t{=}i, U_t{=}u\}, \tag{4}$$

where U_t is the decision variable at time t. Listing the transition probabilities is laborious and cumbersome. Instead, we provide an operational description of the process dynamics and associated probabilities.

Consider an empty system, that is

$$\mathbf{s}_{0,c} = [c, \mathbf{0}, \mathbf{0}, \mathbf{0}], \tag{5}$$

where $\mathbf{0}$ are zero-vectors of an appropriate size. In this state, then, all the buffers are empty, and the channel is in state c. Intuitively, the next "event" driving the system in a different state can be either (*i*) the channel state changes, or (*ii*) a new image arrives. The probability of event (*i*) being the first to happen is simply $\omega/(\omega+\lambda)$. Then the probability that the process transitions from $\mathbf{s}_{0,c}$ to $\mathbf{s}_{0,c'}$ is equal to $\phi_{cc'}\omega/(\omega{+}\lambda)$. Note that u is irrelevant in this case. The probability that the next event is (*ii*) is $\lambda/(\omega+\lambda)$. In this case, the decision variable determines the next state, and we have the following transitions with probability $\lambda/(\omega+\lambda)$:

$$\mathbf{s}_{0,c}{\rightarrow}[c, [0, \ldots, 0, \text{full}], \mathbf{0}, \mathbf{0}] \quad \text{if} \quad u = \text{lc}, \tag{6}$$

$$\mathbf{s}_{0,c}{\rightarrow}[c, \mathbf{0}, [0, \ldots, 0, \text{full}], \mathbf{0}] \quad \text{if} \quad u = \text{ec}, \tag{7}$$

$$\mathbf{s}_{0,c}{\rightarrow}[c, [0, \ldots, 0, \text{split}], \mathbf{0}, \mathbf{0}] \quad \text{if} \quad u = \text{split}. \tag{8}$$

Thus, if $u{=}$lc a full size image is sent to the MD's task buffer for full processing (note that the full DNN model is immediately executed locally as the task is in first position), if $u{=}$ec a full size image is sent to the communication buffer (and transmission immediately begins), and if $u{=}$split then the full image is sent to the MD's task buffer and the head model is used to generate a tensor. From $\mathbf{s}_{0,c}$, all the other transitions have a probability equal to zero. Consider a state $\mathbf{s} = [c, \mathbf{b}_{\text{md}}, \mathbf{b}_{\text{comm}}, \mathbf{b}_{\text{es}}]$, where $0{<}Z_{\text{md}}(\mathbf{s}){<}N_{\text{md}}$, $0{<}Z_{\text{comm}}(\mathbf{s}){<}N_{\text{comm}}$, and $0{<}Z_{\text{es}}(\mathbf{s}){<}N_{\text{es}}$. Thus, all the buffers are non-empty, but also non-full.

From $\mathbf{s}$, the following events might be the first to occur: (*i*) a new image arrives; (*ii*) a task is completed in the MD task buffer; (*iii*) a task is completed in the ES task buffer; (*iv*) the transmission of a data chunk from the communication buffer is completed; and (*v*) the channel changes its state. As expected, when (*i*) to (*iv*) occurs, the task/data chunk is removed from the buffer, and either sent to the next buffer or removed from the system. Event (*v*) is different, in that the state vector remains the same excluding (possibly) the channel state variable c.

Again, the probability that the first event is a specific one in the set of the five possible is the probability that the corresponding exponential variable controlling the time is the smallest in the set. We, then, define the variables

$$\gamma = \gamma_{\text{full}}\mathbf{1}(F_{\text{md}}(\mathbf{z}) = \text{full}){+}\gamma_{\text{head}}\mathbf{1}(F_{\text{md}}(\mathbf{z}) = \text{split}) \tag{9}$$

$$\rho = \rho_{\text{full}}\mathbf{1}(F_{\text{es}}(\mathbf{z}) = \text{full}){+}\rho_{\text{tail}}\mathbf{1}(F_{\text{es}}(\mathbf{z}) = \text{split}) \tag{10}$$

$$\nu = \nu_{c,\text{in}}\mathbf{1}(F_{\text{comm}}(\mathbf{z}) = \text{full}){+}\nu_{c,\text{head}}\mathbf{1}(F_{\text{comm}}(\mathbf{z}) = \text{split}) \tag{11}$$

where $\mathbf{1}(x)$ is the indicator function of event A. We remark that λ and ω are the parameters of the variables determining image arrival and channel state change frequency, respectively. We also define $\mu{=}\lambda + \omega + \gamma + \rho + \nu$.

The probability that the first event from $\mathbf{z}$ is (*i*), (*ii*), (*iii*), (*iv*) or (*v*) is λ/μ, γ/μ, ρ/μ, ν/μ and ω/μ, respectively. Given the occurrence of any of these specific events, the next state is deterministic irrespective of the decision variable u, excluding in (*v*), where the channel transition statistics determine the next state.

Let's define the state vector $\mathbf{s}' = [c', \mathbf{b}'_{\text{md}}, \mathbf{b}'_{\text{comm}}, \mathbf{b}'_{\text{es}}]$. Given the occurrence of (*v*), the state remains the same excluding the channel component, that is, the state transitions to $\mathbf{s}' = [c', \mathbf{b}_{\text{md}}, \mathbf{b}_{\text{comm}}, \mathbf{b}_{\text{es}}]$ with probability $\phi_{cc'}$.

Let's now look at the individual state vectors:

$$\mathbf{b}_{\text{md}} = [0 \ldots 0 \;\; a_{k_{\text{md}}} \ldots a_1], \tag{12}$$

$$\mathbf{b}_{\text{comm}} = [0 \ldots 0 \;\; a_{k_{\text{comm}}} \ldots a_1], \tag{13}$$

$$\mathbf{b}_{\text{es}} = [0 \ldots 0 \;\; a_{k_{\text{es}}} \ldots a_1], \tag{14}$$

where $k_{\text{md}}{=}N_{\text{md}}{-}Z_{\text{md}}(\mathbf{z})$, $k_{\text{es}}{=}N_{\text{es}}{-}Z_{\text{es}}(\mathbf{z})$, and $k_{\text{comm}}{=}N_{\text{comm}}{-}Z_{\text{comm}}(\mathbf{z})$. Given the occurrence of (*i*), the process transitions to the following states

$$\mathbf{s}{\rightarrow}[c \; [0 \ldots \text{ full } a_{k_{\text{md}}} \ldots a_1] \; \mathbf{b}_{\text{comm}} \; \mathbf{b}_{\text{es}}] \quad \text{if} \quad u{=}\text{lc}, \tag{15}$$

$$\mathbf{s}{\rightarrow}[c \; \mathbf{b}_{\text{md}} \; [0 \ldots \text{ full } a_{k_{\text{md}}} \ldots a_1] \; \mathbf{b}_{\text{es}}] \quad \text{if} \quad u{=}\text{es}, \tag{16}$$

$$\mathbf{s}{\rightarrow}[c \; \mathbf{b}_{\text{md}} \; [0 \ldots \text{ split } a_{k_{\text{md}}} \ldots a_1] \; \mathbf{b}_{\text{es}}] \quad \text{if} \quad u{=}\text{split}, \tag{17}$$

with probability one.

If (*ii*) occurs and the first element of $\mathbf{b}_{md}$ is equal to **full** then the vector shifts right (removing the first element) and a zero is appended. If the first element of $\mathbf{b}_{\text{md}}$ is equal to **split**, then that first element is moved to replace a zero in $\mathbf{b}_{\text{comm}}$, the vector $\mathbf{b}_{\text{md}}$ shifts right (removing the first element) and a zero is appended. If (*iii*) occurs, then the vector $\mathbf{b}_{\text{es}}$ shifts

right (removing the first element) and a zero is appended. If (*iv*) occurs, then the first element of the $\mathbf{b}_{\text{comm}}$ is moved to replace a zero in $\mathbf{b}_{\text{es}}$, the vector $\mathbf{b}_{\text{comm}}$ shifts right (removing the first element) and a zero is appended.

If one or more buffers are empty, then the corresponding event has probability equal to zero, and the respective rates are removed from the denominator of the transition probabilities. If one or more buffer is full, then the transitions described above need to be modified to account for the fact that tasks/data cannot be moved to them and are, instead, erased.

V. Optimal Policy

The transition probabilities described in the previous section are conditioned on the action u chosen by the selector. We define, then, the policy ξ guiding such choice.

A. Performance Metrics and Cost Functions

We are interested in two key metrics: (*a*) the average time lapse between task arrival and departure from the system (total inference time) and (*b*) the task loss rate, that is, the probability that a task is sent to a full buffer. Given the complexity of the system, the definition of such metrics is non-trivial. Due to space constraints, we again provide an operational description that allows the derivation of the metrics.

Given the nature of the process, for each metric $r_i : \mathcal{S} \times \mathcal{S} \times \mathcal{U} \to \mathbb{R}$, that assigns a cost to the tuple (state, state, action), we associate the time-average of the metric r_i defined as

$$\bar{r_i} = \lim_{n \to +\infty} \frac{1}{n} \sum_{k=1}^{n} E\left[r_i(s_k, s'_k, u_k)\right]. \tag{18}$$

We now express the quantities needed to compute the performance metrics listed above. The average number of tasks successfully completed is the time average of the cost function:

$$r_1(s_k, s'_k, u_k) = \begin{cases} 1 & \text{if task successful at MD or ES} \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

Similarly we denote $r_2(s_k, s'_k, u_k)$ the average number of tasks discarded and $r_3(s_k, s'_k, u_k)$ the time passed.

The average total inference time is the ratio between the accumulated delay of all the images and the number of images whose analysis is completed. Therefore we can express it as

$$\frac{\bar{r}_3}{\bar{r}_1} = \frac{\lim_{n\to+\infty} \sum_{k=1}^{n} E[r_3(s_k, s'_k, u_k)]}{\lim_{n\to+\infty} \sum_{k=1}^{n} E[r_1(s_k, s'_k, u_k)]}. \tag{20}$$

B. Optimization Problem

Let us introduce $\hat{\mathbf{U}}$, the sequence of actions that minimizes the objective $R(\mathbf{U})$ over the space U^{∞} of all possible infinite action sequences under M_c linear constraints. Then we can express the problem as a Linear Fractional Program:

$$\begin{aligned} \hat{\mathbf{U}} = \arg\inf_{\xi} & \frac{\sum_{s\in\mathcal{S}} \pi_\xi(s) r_3(s,\xi)}{\sum_{s\in\mathcal{S}} \pi_\xi(s) r_1(s,\xi)} \\ s.t. \quad & \beta_q \frac{\sum_{s\in\mathcal{S}} \pi_\xi(s) r_{c_n(q)}(s,\xi)}{\sum_{s\in\mathcal{S}} \pi_\xi(s) r_{c_d(q)}(s,\xi)} + \lambda_q \leq \gamma_q \end{aligned} \tag{21}$$

where $q = 1, 2, ..., M_c$ enumerates the program constraints and $c_n(q), c_d(q)$ are indexes for the necessary cost functions appearing in the constraints. The problem stated above has multiple dependencies between the randomized stationary policy $\xi(s,u)$: $\pi_\xi(s)$, the stationary distribution of being in a given state, and the cost $r_i(s, \xi(s,u))$.

For this reason we define $\boldsymbol{\kappa} = g\boldsymbol{\omega} = g\pi_\xi(s)\xi(s,u)$, where the intermediate variables ω can be interpreted as the probability that the process is in state s and action u is taken. The change of variables $\boldsymbol{\kappa} = g\boldsymbol{\omega}$ is necessary to scale the magnitude of probabilities to the one of the cost functions, reaching the final problem formulation:

$$\begin{aligned} \{\hat{\boldsymbol{\kappa}}, \hat{g}\} = \; & arg\min_{\boldsymbol{\kappa}, g} r_3^T \boldsymbol{\kappa} \\ s.t. \quad & (\beta r_{c_n} + (\lambda - \gamma) r_{c_d})^T \boldsymbol{\kappa} \leq \mathbf{0}_{M_c,1} \\ & \mathbf{1}_{1,A}\boldsymbol{\kappa} - g = 0 \\ & r_{c_d}^T \boldsymbol{\kappa} = 1 \\ & \mathbf{P}\boldsymbol{\kappa} = \mathbf{0}_{|\mathcal{S}|,1} \\ & g \geq 0, \kappa_{s,u} \geq 0 \quad \forall s \in \mathcal{S}, \forall u \in \mathcal{U}, \end{aligned} \tag{22}$$

where $A = |\mathcal{S} \times \mathcal{U}|$ is the cardinality of the state-action space, and where $\mathbf{0}_{m,n}, \mathbf{1}_{m,n}$ are matrices with m rows and n columns, with all elements equal to 0 and 1 respectively.

C. Optimal Policy

As demonstrated in [15], if the Markov process is *unichain*, that is, the whole state space is a single recurrent class, then at least one optimal memoryless stationary policy exists in the form of the following conditional probability

$$\xi(\mathbf{s}, u) = \Pr\{U_t = u \mid S_t = \mathbf{s}\}. \tag{23}$$

We use the problem formulated above to find the policy ξ through direct computation using the Linear Fractional Problem defined in the previous section. Applying the Simplex Algorithm to the problem in Eq. (22) we find $\kappa_{s,u}$, from which we can extract the policy using the following formulas:

$$\hat{\xi}(s,u) = \frac{\kappa(s,u)}{\sum_{u\in\mathcal{U}} \kappa(s,u)} \tag{24}$$

where if $\kappa(s,u) = 0$ the state is transient under the optimal policy, making $\hat{\xi}(s,u) = 1$ for one random action and $\hat{\xi}(s,u) = 0$ otherwise.

VI. Results

In this section, we provide results discussing the performance of the dynamic selection we proposed with respect to edge computing. In order to find the parameters to use in solving LFP, we measured the inference time of DenseNet-169 over a large image classification dataset, on two devices: a Raspberry Pi 3b+ and Nvidia Jetson TX2. We created a split architecture using the head distillation technique described in [5] with a similarly shaped bottleneck. Note that we selected the smallest bottleneck size that preserved accuracy. The inference time and size of the data to be transmitted over the network are reported in Table II.

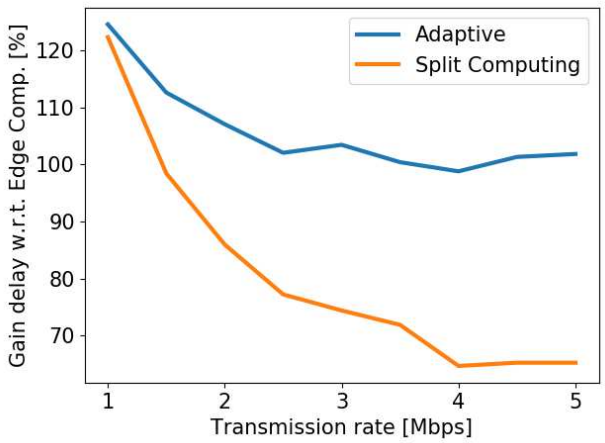

Figure 2: Delay ratio w.r.t. edge computing as a function of transmission rate.

Table II: Inference time and size of data to be transferred

	Mobile Device	Edge Server	Data
Full comp.	7.4 s	0.3 s	600 Kb
Split comp.	0.44 s	0.25 s	85 Kb

Fig. 2 shows the ratio between the delay incurred using a fixed edge computing policy and the split computing. We remark that both adaptive and Split Computing outperform simple task offloading when transmission rates are below 2 Mbps. From the plot, it is clear that edge computing is the best strategy when data rates are sufficiently large (over 10 Mbps in the considered setting). The optimal policy implements a hybrid behavior. Whe the transmission rate is small, it mainly uses split computing. As the transmission rate increases, edge computing is used more often.

In Fig. 3, we plot the tradeoff between the number of successful tasks and the average delay. It can be observed how not only the delay is reduced using an adaptive approach, but also the number of successfully processed tasks increases. This is again due to the flexibility that split computing offers in a range of data rates where edge offloading is simply unusable due to the size of the full frame.

VII. Conclusions

In this paper we leveraged head network distillation technique, using an adaptable logic, reinterpreting edge offloading, generally thought as a binary choice. We presented a detailed stochastic model for such scenarios using queuing theory, formulated an optimization problem, and solved it using Linear Fractional Programming. Results show a clear improvement for asymmetric systems with impaired communications.

Acknowledgment

This work was supported by the NSF grant IIS-1724331 and MLWiNS-2003237.

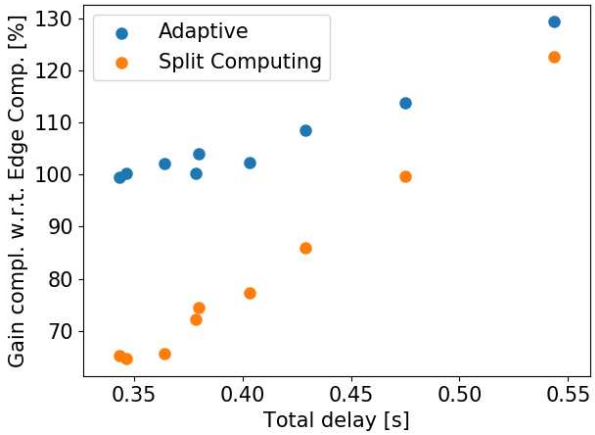

Figure 3: Gain in task computed w.r.t. edge computing, as a function of the total inference delay.

References

[1] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[2] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *Proceedings of IEEE INFOCOM 2013*, 2013, pp. 1285–1293.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of ACM ASPLOS*. New York, NY, USA: ACM, 2017, pp. 615–629.

[4] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

[5] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled split deep neural networks for edge-assisted real-time systems," in *Proceedings of the 2019 MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2019, pp. 21–26.

[6] Y. Matsubara and M. Levorato, "Split computing for complex object detectors: Challenges and preliminary results," *arXiv preprint arXiv:2007.13312*, 2020.

[7] Y. Matsubara and M. Levorato, "Neural compression and filtering for edge-assisted real-time object detection in challenged networks," *arXiv preprint arXiv:2007.15818*, 2020.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Fourth International Conference on Learning Representations*, 2016.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Fourth International Conference on Learning Representations*, 2016.

[11] H.-J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon, "Computation offloading for machine learning web apps in the edge server environment," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1492–1499.

[12] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, vol. 1, no. 2, 2017, p. 3.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

[15] K. W. Ross, "Randomized and past-dependent policies for markov decision processes with multiple constraints," *Operations Research*, vol. 37, no. 3, 1989.